UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 09/662,399 | 09/14/2000 | Nagender P. Vedula | MS147164.1 | 1314 |

27195 7590 05/03/2006

AMIN & TUROCY, LLP
24TH FLOOR, NATIONAL CITY CENTER
1900 EAST NINTH STREET
CLEVELAND, OH 44114

| EXAMINER | |
|---|---|
| BASOM, BLAINE T | |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2173 | |

DATE MAILED: 05/03/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

| Office Action Summary | Application No. | Applicant(s) | |
|---|---|---|---|
| | 09/662,399 | VEDULA ET AL. | |
| | Examiner | Art Unit | |
| | Blaine Basom | 2173 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☐ Responsive to communication(s) filed on *10 February 2006*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) *1-45* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-45* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 September 2000* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

## DETAILED ACTION

This Office action is responsive to the Request for Continued Examination (RCE) filed under 37 CFR §1.53(d) for the instant application on 2/10/2006. The Applicants have properly set forth the RCE, which has been entered into the application, and an examination on the merits follows herewith.

### *Response to Arguments*

The Examiner acknowledges the Applicants' amendments to claims 1-45. Regarding these claims, the Applicants argue that Oppenheim (U.S. Patent No. 5,734,905), the COM specification (Microsoft's "Component Object Model Specification"), Jordan (U.S. Patent No. 5,778,227), and Faustini (U.S. Patent No. 6,496,870), described in the previous Office Action, fail to teach or suggest a source object node which is one of a plurality of hierarchically displayed nodes, and a target object node which is one of a plurality of hierarchically displayed nodes, as is claimed. In response, the Examiner presents the "The Macintosh Bible, 7th Edition," which as shown below, teaches hierarchically arranging and displaying such nodes. The Applicants' arguments have thus been considered, but are moot in view of the following new grounds of rejection.

***Claim Rejections - 35 USC § 103***

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 16, 23, 30, 32-35, 37, 38, 41, 42, 43, and 45 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,734,905, which is attributed to Oppenheim, and also over the Mac OS 8.5 operating system, as is described by "The Macintosh Bible, 7th Edition" (hereafter referred to as "the Macintosh Bible"), written by Sharon Zardetto Aker. In the context of object oriented design, Oppenheim discloses a "transformer object," which is used to perform an action on a "transformee object" so as to produce a "transformed object" (see column 3, lines 43-53). A transformer object may also be linked between a first and a second object so that data output from the first object is processed by the transformer object, and then input into the second object. For example, figure 8 of Oppenheim shows data output from an "A/D Converter" object, wherein this data is then modified by a transformer object, referred to in the drawing as a "Signal Processor," and then input into a "Filter" object (see column 8, lines 23-65). In this case, the transformer object provides a mapping between the A/D Converter object, i.e. source object, and the Filter object, i.e. target object. The transformer object taught by Oppenheim is therefore considered a "function object," and any computer or application implementing this transformer object is considered a "mapping tool."

Specifically regarding claim 16, Oppenheim discloses that the transformer object includes a set of transformation scripts (see column 5, lines 38-41), which are used to modify a

transformee object (see column 5, lines 45-50). The transformer object thus includes a "script component" as recited in claim 16. Additionally, Oppenheim discloses that each transformer object has an associated image which can be displayed (see column 3, lines 37–39). As shown in figure 8, this image includes an input and an output which allow a user to graphically associate the input with a source object and to associate the output with a target object (see column 8, lines 37-65). The transformer object disclosed by Oppenheim thus includes a "graphical component" as recited in claim 16. Moreover, Oppenheim notes that each transformer object has an associated interface, which is used to communicate with other objects (see column 5, lines 30-45). Consequently, the transformer object includes an "interface component" as recited in claim 16. The image associated with the transformer object, which corresponds to the "graphical component" recited in claim 16, is displayed in a graphical user interface (see column 3, lines 32-38). Oppenheim also teaches, as expressed in claim 16, graphically associating a source object node with an input of the transformer object using a user selection device. This is specifically done by dragging and dropping the source object on the input of the transformer object (see column 8, lines 37-53). Similarly, Oppenheim teaches the idea for graphically associating a target object node with an output of the transformer object, wherein this is specifically done by dragging and dropping the target object on the output of the transformer object (see column 8, lines 54-57). It is understood that a mapping is then created between the source object, the transformer object, and the target object, such that the function of the transformer object is performed according to data from the source object node, and the transformer object provides an output to the target object according to this function (see column 8, lines 60-65). Oppenheim, however, does not explicitly disclose that the source object node is

displayed hierarchically amongst a plurality of source object nodes and that the target object node displayed hierarchically amongst a plurality of target object nodes, as is claimed. Nevertheless, hierarchically organizing and displaying such objects is well known in the art.

For example, in Mac OS 8.5, icons representing programs or program data are accessible through a hierarchical arrangement of folders (for example, see "Working with Desktop Icons" on pages 8-10 of the Macintosh Bible; and "Working with Windows" on pages 11-14 of the Macintosh Bible).

It would have consequently been obvious to one of ordinary skill in the art, having the teachings of Oppenheim and Mac OS 8.5 before him at the time the invention was made, to modify the source and target objects taught by Oppenheim such that they are hierarchically organized and displayed, as is done in Mac OS 8.5. It would have been advantageous to one of ordinary skill to utilize this combination, because such a hierarchical arrangement facilitates identification of objects of interest to the user, as is demonstrated by Mac OS 8.5. Oppenheim and Mac OS 8.5 thus teach graphically associating a source object node with an input of the transformer object, and graphically associating a target object node with an output of the transformer object, wherein whereby the source and target objects, and thus their associated nodes, are hierarchically displayed.

Regarding claim 23, Oppenheim notes that transformer objects may be capable of performing a plurality of functions on a transformee object (see column 3, lines 55-59). Since, as described above in the rejection for claim 16, a set of transformation scripts is included in the interface of a transformer object and is used to perform a function, it is interpreted that a transformer object capable of a plurality of functions would have a plurality of sets of scripts,

where each set of scripts is used to perform a function. Thus the transformer object taught by Oppenheim includes a plurality of script components, as recited in claim 23. Furthermore, it is understood that a different image may be associated with each of the plurality of functions of a transformer object. In figure 8, the graphical image associated with an object that performs an A/D conversion function is a box with the text "A/D Converter," reference number 270. On the other hand, the graphical image associated with an object that performs a signal processing function is a box with the text "Signal Processor," reference number 272. It is therefore interpreted that a transformer object capable of a plurality of functions would have a plurality of images, each associated with a function. As shown above in the rejection for claim 16, each image may include an input and an output which allow a user to graphically associate the input with a source object and to associate the output with a target object. The transformer object taught by Oppenheim thus includes a plurality of graphical components, as recited in claim 23. Also, as described above in the rejection for claim 16, Oppenheim notes that each transformer object has an associated interface, which is used to communicate with other objects (see column 5, lines 30-45). It is interpreted that this interface is associated with the plurality of scripts and images described above. Lastly, Oppenheim discloses that one of the images is displayed in a graphical user interface (see column 3, lines 36-38).

In reference to claim 30, the rejection for claim 16 above shows that the interfaces of the transformer objects disclosed by Oppenheim are used to provide the computer executable instructions, i.e. scripts, of the transformer objects to a compiler. It is also noted that the computer or application program implementing the transformer objects disclosed by Oppenheim is considered a mapping tool for the reasons described in the rejection for claim 16. Oppenheim

however does not explicitly state that such a mapping tool includes a compiler component, wherein this compiler component is invoked to generate compiled output code. Nevertheless, it is understood that for the transformer objects to modify source or targets objects as described above, a compiler component is inherently necessary, wherein this compiler generates compiled output code.[1]

As for claim 32, the computer or application program implementing the transformer objects disclosed by Oppenheim, is considered a mapping tool for the reasons described in the rejection for claim 16. Consequently, such a mapping tool is understood to include a means for providing a function object having: a script component with computer-executable instructions for performing a function, a graphical component associated with the function and having an input and an output, and an interface component. Moreover, since Oppenheim discloses that the images associated with transformer objects are displayed (see column 3, lines 36-38), it is understood that this mapping tool includes a means for displaying the graphical component in the user interface, as is recited in claim 32. Oppenheim also teaches that the mapping tool includes a user selection device for graphically associating a source object node with an input of the transformer object, and for graphically associating the target object node with the output. This is specifically done by dragging and dropping the source object on the input of the transformer object, and by dragging and dropping the target object on the output of the transformer object (see column 8, lines 37-57). Mac OS 8.5 further teaches that such source and target objects may be hierarchically displayed, as is described above. Accordingly, the mapping tool of Oppenheim

[1] For example, U.S. Patent No. 5,778,227, which is attributed to Jordan and described subsequently in the rejection for claim 28, presents objects like those presented by Oppenheim. These objects may be modified, wherein a compiler is necessary to implement these modifications (see column 4, lines 23-52).

and Mac OS 8.5 is additionally understood to comprise a means for displaying a source object and a target object in the graphical user interface, the source object (e.g. a directory) comprising a plurality of hierarchically arranged nodes and the target object (e.g. a second directory) comprising a plurality of hierarchically arranged nodes. Lastly, as was shown above in the rejection for claim 30, the mapping tool taught by Oppenheim necessarily includes a compiler. It is understood that such a compiler can create a mapping between the source object, the transformer object, and the target object, such that the function of the transformer object is performed according to data from the source object node, and the transformer object provides an output to the target object according to this function (see column 8, lines 60-65). Thus the mapping tool of Oppenheim and Mac OS 8.5 is further considered to comprise a means for creating a mapping, like recited in claim 32.

As for claim 33, the computer system taught by Oppenheim, which is considered a mapping tool for the reasons shown above in the rejection for claim 16, provides a user the ability to create objects. For example, Oppenheim discloses that:

> *...in a first object transformation, a first object operator 242 is used to create another object operator 244 by "extracting" a desired characteristic or set of characteristics from an object 246 that contains an example of something the user would like to replicate in other objects. For instance, operator object 242 might contain software for extracting the "style" of any specified document. By "slapping" this operator object 242 on an object 246 that contains a document, a new operator object 244 is created that can used to reformat other documents with the extracted style.* (See column 7, lines 33-43).

Thus Oppenheim discloses that a new object may be created by dragging an already existing object over a second already existing object, wherein it is understood that this new object may be a transformer object like that discussed above in the rejection for claim 16. Because such a transformer object includes a script component, graphical component, and an interface component, as was shown above in the rejection for claim 16, it is also understood that creating a transformer object necessarily results in creating a script component, creating a graphical component, creating an interface component, and associating the script component, the graphical component, and the interface component, as is expressed in claim 33. In addition to the above-described method of dragging and dropping an existing object onto a second existing object, Oppenheim discloses that a user may also create a transformer object by using various programs (see column 7, lines 1-20). It is interpreted that these programs are used to create a script component, create a graphical component, create an interface component, and associate the script component, the graphical component, and the interface component, as is expressed in claim 33. As further above, Oppenheim discloses that the graphical component may comprise an input configured to be associated with a node of a source object, and an output configured to be associated with a node of a target object. Mac OS 8.5 further teaches that such source and target objects may be hierarchically displayed, as is described above. Accordingly, it is understood that the graphical user interface of the function object may comprise an input configured to be associated with a node of a source object, wherein the source object (e.g. a directory) conforms to a first schema, and an output configured to be associated with a node of a target object, wherein the target object (e.g. a second directory) conforms to a second schema.

With respect to claims 34, 35, 37, and 38 it was shown above in the rejection for claim 33 that Oppenheim discloses that a user may create a transformer object by using various programs. Moreover, it is understood that a user uses such a program to create text files, which contain lines of code, as Oppenheim explicitly states:

> *...the transformer object 230 may contain a computer program, created using a program such as Small Talk, that can be used to modify the musical score in object 232. For instance, a simple program containing one line of code:*
>
> *event velocity: event pitch*
>
> *would cause the loudness (called velocity in the MIDI language used to control music synthesizers) of each musical note in the score to be modified so as to be proportional to its pitch.* (See column 7, lines 12-20).

It is therefore understood that creating a script component for a transformer object with such a program involves receiving a user-defined text file including computer-executable instructions, wherein the script component is created using the computer-executable instructions from the text file. As for claim 37, since this script component disclosed by Oppenheim is part of the interface of a transformer object (see column 5, lines 38-41), it is interpreted that the interface of a transformer object is created according to the information received in the text file described above. Lastly, it is understood that such a text file may include computer-executable instructions in basic, visual basic, VB script, C++, visual C++, java, java script, Perl, and XML, which are all well-known in the art.

In regard to claim 41, it was shown above in the rejection for claim 33 that the transformer objects disclosed by Oppenheim may be created by: creating a script component

having computer-executable instructions for performing a function using the user interface; creating a graphical component associated with the function and having an input and an output; creating an interface component adapted to provide the script component to a compiler in the mapping tool and to provide the graphical component to the graphical user interface; and associating the script component, the graphical component, and the interface component. As further above, Oppenheim discloses that the input of the graphical component may be associated with a node of a source object, and that the output may be associated with a node of a target object. Mac OS 8.5 further teaches that such source and target objects may be hierarchically arranged and displayed, as is described above. Accordingly, it is understood that the graphical component of the function object may comprise an input configured to be associated with a source object node from amongst a hierarchical arrangement of source object nodes, and an output configured to be associated with target object node from amongst a hierarchical arrangement of target object nodes. It is therefore understood that a computer implementing the function objects taught by Oppenheim, as modified by Mac OS 8.5, provides a means for creating this script component, creating the graphical component, creating the interface component, and associating the script component, the graphical component, and interface component.

Regarding claim 42, the method described above in the rejection for claim 16, may be implemented via computer memory (see column 4, lines 53-63). It is therefore understood that such a memory, storing this method, is considered to be a computer readable medium equivalent to that recited in claim 42.

Regarding claims 43 and 45, the method described above in the rejection for claim 33, may be implemented via computer memory (see column 4, lines 53-63). It is therefore understood that such a memory, storing this method, is considered to be a computer readable medium equivalent to that recited in both claims 43 and 45.

Claims 1-15, 17-22, 24-27, and 44 are rejected under 35 U.S.C. 103(a) as being unpatentable over the combination of Oppenheim and Mac OS 8.5, which is described above, and also over Microsoft's "Component Object Model Specification" (which is hereafter referred to as the "COM specification"). Regarding claims 1 and 44, the transformer object presented by Oppenheim includes a set of transformation scripts (see column 5, lines 38-41), which are used to modify a transformee object (see column 5, lines 45-50). The transformer object thus includes a "script component" as recited in both claims 1 and 44. Additionally, Oppenheim discloses that each transformer object has an associated image which can be displayed (see column 3, lines 37–39). As shown in figure 8, this image includes an input and an output which allow a user to graphically associate the input with a source object and to associate the output with a target object (see column 8, lines 37-65). Mac OS 8.5 further teaches that such objects may be hierarchically arranged and displayed, as is described above. Accordingly, it is understood that the image of the function object may comprise an input configured to be associated with a source object node from amongst a plurality of hierarchically displayed nodes, and an output configured to be associated with target object node from amongst a plurality of hierarchical displayed nodes. The transformer object disclosed by Oppenheim, as modified by Mac OS 8.5, thus includes a "graphical component" as recited in both claims 1 and 44. Lastly, Oppenheim notes that each

transformer object has an associated interface, which is used to communicate with other objects (see column 5, lines 30-45). It is interpreted that since, as shown above, each transformer object includes scripts, which must generally be compiled, and since each of the transformer objects have associated images that are displayed in a graphical user interface, the transformer objects disclosed by Oppenheim must include an interface component adapted to provide the scripts to a compiler and to provide the images to the graphical user interface. Neither Oppenheim nor Mac OS 8.5, however, explicitly teach that such an interface component has a globally unique identifier, as is recited in both claims 1 and 44.

> The COM specification presents the Component Object Model, which is described as: *...an object-based programming model designed to promote software interoperability; that is, to allow two or more applications or 'components' to easily cooperate with one another, even if they were written by different vendors at different times, in different programming languages, or if they are running on different machines running different operating systems.* (see page 5 of Chapter 1).

The COM specification further states that to provide such interoperability, COM defines and implements mechanisms that allow applications to connect to each other as software objects (see page 5 of Chapter 1). Thus, like Oppenheim, the COM spec concerns the interaction of applications and more particularly, the interaction of objects. It would therefore have been obvious to one of ordinary skill in the art, having the teachings of Oppenheim, Mac OS 8.5, and the COM specification before him at the time the invention was made, to modify the objects taught by the combination of Oppenheim and Mac OS 8.5 such that they are implemented as COM objects, as described in the COM specification. It would have been advantageous to one

of ordinary skill to utilize such a combination because the Component Object Model is a standard means for object interoperability, thus enabling application developers to build and distribute applications more easily, providing vendors with a single model for interacting with other applications, and allowing end-users to see a greater range of software choices, as is taught by the COM specification (see page 5). Regarding claims 1 and 44, each COM object intrinsically includes one or more globally unique identifier in order to identify the one or more interfaces of the object (see pages 9 and 10 of the COM specification).

As for claim 2, the interfaces of each COM object inherently include a category identifier component. For example the COM specification discloses that,

> *... an interface is a strongly-typed group of semantically-related functions, also called "interface member functions." The name of an interface is always prefixed with an "I" by convention, as in* ***IUnknown****. (The real identity of an interface is given by its GUID; names are a programming convenience, and the COM system itself used the GUIDs exclusively when operating on interfaces.) In addition, while the interface has a specific name (or type) and names of member functions,* ***it defines only how one would use that interface and what behavior is expected from an object through that interface****.*
>
> (emphasis added) (See page 13 of Chapter 1).

Thus, the name of an object's interface is considered a category identifier component, as it defines the behavior expected from an object, or more specifically, the behavior expected from an interface of an object.

With respect to claim 3, the COM specification discloses that a globally unique identifier of a COM object interface is used to "identify object classes and the interfaces those objects may

support" (see page 9 of Chapter 1). An interface's globally unique identifier is thus considered a class identifier component, because it identifies object classes.

In regard to claims 4 and 5, the name of an object's interface describes the behavior expected from an object through that interface, as is shown above in the rejection for claim 2. Since an interface is a "strongly-typed group of semantically-related functions" (see page 13 of Chapter 1), the name of an object's interface is also considered to identify the behavior of the group of functions comprising the interface. Therefore, the name of an object's interface is also understood to be a function category identifier component. Oppenheim discloses that a transformer object may implement, among others, mathematical and conversion functions (see column 7, lines 1-5 and column 8, lines 37-40). Therefore, it is understood that such a transformer object, implemented as a COM object, may have an interface with the name of mathematical or conversion, since that describes the behavior expected from the functions of the interface. The function category identifier component may thus be mathematical or conversion, which are two of string, mathematical, logical, data, conversion, scientific, advanced, and custom.

Referring to claims 6 and 7, the COM specification teaches that drag and drop ability is provided by means of specific interfaces on COM objects (see page 26 of Chapter 1). Oppenheim discloses that the image associated with a transformer object can be dragged and dropped (see column 6, lines 15-26). It is therefore understood that a transformer object, implemented as a COM object, would comprise a specific interface, i.e. a second interface, adapted to allow a user to drag and drop the image in a graphical user interface. It is also

understood that this second interface is associated with the function object by any client program using the transformer object, such as a mapping tool.

Regarding claim 8, Oppenheim notes that transformer objects may be capable of performing a plurality of functions on a transformee object (see column 3, lines 55-59). Since, as described above in the rejection for claim 1, a set of transformation scripts is included in the interface of a transformer object and is used to perform a function, it is interpreted that a transformer object capable of a plurality of functions would have a plurality of sets of scripts, where each set of scripts is used to perform a function. Thus the transformer object taught by Oppenheim includes a plurality of script components, as recited in claim 8. Furthermore, it is understood that a different image may be associated with each of the plurality of functions of a transformer object. In figure 8, the graphical image associated with an object that performs an A/D conversion function is a box with the text "A/D Converter," reference number 270. On the other hand, the graphical image associated with an object that performs a signal processing function is a box with the text "Signal Processor," reference number 272. It is therefore interpreted that a transformer object capable of a plurality of functions would have a plurality of images, each associated with a function. As shown above in the rejection for claim 1, each image may include an input and an output which allow a user to graphically associate the input with a source object and to associate the output with a target object. The transformer object taught by Oppenheim thus includes a plurality of graphical components, as recited in claim 8. By the same reasoning described above in the rejection for claim 1, it is interpreted that since a transformer object capable of a plurality of functions includes a plurality of scripts, which must generally be compiled, and since this transformer object has a plurality of associated images that

may be displayed in a graphical user interface, such a transformer object must include an interface component adapted to provide the plurality of scripts to a compiler and to provide the plurality of images to a graphical user interface.

Regarding claims 9 and 10, the name of a COM object's interface is also understood to be a function category identifier component, for the reasons described above in the rejection for claims 4 and 5. A COM object may a have a plurality of interfaces (see page 14 of Chapter 1). Therefore, a COM object may have a plurality of function category identifier components, each associated with the functions comprising its interface. Therefore, the transformer object capable of a plurality of functions as described by Oppenheim, and implemented as a COM object, is considered to comprise a plurality of function category identifier components, each associated with a plurality of the functions. And for the reasons described above in the rejection for claims 4 and 5, the function category identifier component may be one of string, mathematical, logical, data, conversion, scientific, advanced, and custom.

With respect to claims 11-14, it is understood that since they are described as "scripts," the plurality of script components expressed by Oppenheim are in a scripting language. It is further interpreted that this scripting language may be one of java script, visual basic script, and visual C++, which are all well-known scripting languages.

Regarding claim 15, it would have been obvious, due to the reasons presented in claim 1, to implement the transformer object disclosed by Oppenheim as a COM object, wherein the interface component would be a COM interface.

In reference to claims 17-19, it was shown above that the manner in which transformer objects are presented and used, as described by Oppenheim and Mac OS 8.5, teaches the method

recited in claim 16 for creating a mapping between a source object having a source object node and a target object having a target object node. Neither Oppenheim nor Mac OS 8.5, however, explicitly teaches that such a transformer object is obtained from a host computer, as is recited in claim 17. Consequently, Oppenheim and Mac OS 8.5 do not disclose that that a transformer object may obtained from a DLL file in the host computer, as is expressed in claim 18, or that obtaining the transformer object further includes the steps of searching the DLL files in the host computer; identifying the transformer object in the DLL files; and loading the transformer object into the mapping tool; as is recited in claim 19.

The COM specification defines a client as an application that instantiates and uses objects (see page 20 of Chapter 1). It is understood that such an application encompasses the program described by Oppenheim, which as described above, is considered a mapping tool. According to the COM specification, a client uses a server, which is a piece of code used to implement and provide access to objects (see page 20 of Chapter 1). Such a server may reside in the client's process space. In this case, the server is referred to as an "in-process server" and is implemented as a dynamic link library (DLL) (see page 21 of Chapter 1). It is understood that if a server runs in the client's process space, it generally runs on the same computer that is implementing the client, i.e. application. Therefore, the server runs on the host computer, meaning that the objects that this server provides are obtained from the host computer. And since the server is implemented as a DLL file, the objects are thus obtained from a DLL file in the host computer. It is understood that a client application that requires a plurality of objects, such as that described by Oppenheim, would require either a plurality of servers or a server that can provide a plurality of objects. Consequently, obtaining a COM object necessitates a search to find the appropriate

server, and/or a search by a server to find the appropriate object. Since a server is implemented as a DLL file, this implies searching the DLL files in the host computer. Obtaining the COM object would further require identifying the function object in the searched DLL files and loading the function object. Lastly, for the reasons described above in the rejection for claim 1, it would have been obvious to one of ordinary skill in the art, having the teachings of Oppenheim, Mac OS 8.5, and the COM specification before him at the time the invention was made, to modify the objects taught by Oppenheim and Mac OS 8.5 so that they are implemented as COM objects, meaning that they are provided via a server as described above.

In reference to claims 20-22, it was shown above that the manner in which transformer objects are presented and used, as described by Oppenheim and Mac OS 8.5, teaches the method recited in claim 16 for creating a mapping between a source object having a source object node and a target object having a target object node. Neither Oppenheim nor Mac OS 8.5, however, explicitly teaches that such a transformer object is obtained from a global communications network, as is expressed in claim 20. Consequently, Oppenheim and Mac OS 8.5 fail to teach that a transformer object may obtained from an Internet web site, as is expressed in claim 21, or that obtaining the transformer object further includes the one of the steps of searching the Internet for a transformer object according to a mapping tool startup script; searching for the transformer object according to a user command; and searching a web site provided by the user for the transformer object, as is expressed in claim 22.

The COM specification discloses that:

*Any client can work with any object anywhere else on the machine or network, and because the object reusability mechanisms of containment and aggregation maintain a*

*client/server relationship between objects, reusability is also possible across process and network boundaries.* (See page 11 of Chapter 1).

The COM specification thus teaches that an object may be obtained via a network. It is understood that such a network may be the Internet, which is a well-known and widely used global communications network. It is further understood that the object may be accessed from an Internet web site, provided by the user, by searching the web site, as this is a common mechanism for accessing data via the Internet. Lastly, for the reasons described above in the rejection for claim 1, it would have been obvious to one of ordinary skill in the art, having the teachings of Oppenheim, Mac OS 8.5, and the COM specification before him at the time the invention was made, to modify the objects taught by Oppenheim and Mac OS 8.5 so that they are implemented as COM objects, meaning that they are provided over a network as described above.

As per claim 24, the COM specification teaches that drag and drop ability is provided by means of specific interfaces on COM objects (see page 26 of Chapter 1). Oppenheim discloses that the image associated with a transformer object can be dragged and dropped (see column 6, lines 15-26). It is therefore understood that a transformer object, implemented as a COM object as described above, would comprise a specific interface, i.e. a second interface, adapted to allow a user to drag and drop the image in a graphical user interface. This transformer object, implemented as a COM object, is considered a wrapper object.

In reference to claims 25-27, it was shown above that the manner in which transformer objects are presented and used, as described by Oppenheim and Mac OS 8.5, teaches the method recited in claim 16 for creating a mapping between a source object having a source object node

and a target object having a target object node. Neither Oppenheim nor Mac OS 8.5, however, explicitly teaches that the image associated with the transformer object is obtained from a function object source via its interface, as is expressed in claim 25. Consequently, Oppenheim and Mac OS 8.5 do not disclose that such a function object source is a DLL file in the host computer, as is expressed in claim 26, or that the image is obtained from a DLL file via the Internet, as is expressed in claim 27.

As described above, the COM specification defines a server as a piece of code that implements and provides COM objects. Accordingly, it is understood that any data associated with a COM object, such as an image, is obtained from a server. Moreover, the COM spec defines a COM object interface as a "strongly-typed group of semantically-related functions" and notes that "clients always access the object through clearly defined contracts: the interfaces that the object supports" (see page 13 of Chapter 1). It is therefore understood that for a client application to obtain data from a COM object, such as an image associated with the object, it is necessary to use an interface function of the object. Consequently, it is understood that with the transformer object disclosed by Oppenheim, implemented as a COM object as described above, displaying an image associated with an object as Oppenheim describes requires obtaining the image associated with the object from a function object source, i.e. server, via the interface of the object. Specifically regarding claim 26, the COM specification states that this function object source, i.e. server, may reside in the client's process space. In this case, the server is referred to as an "in-process server" and is implemented as a dynamic link library (DLL) file (see page 21 of Chapter 1). It is understood that if a server runs in the client's process space, it generally runs on the same computer that is implementing the client, i.e. application. Therefore, the server runs

on the host computer, meaning that the function object source is a DLL file on a host computer. As for claim 27, the COM specification also states that the server may reside on a remote computer and be implemented as a DLL file (see page 21 of Chapter 1). It is consequently understood that, in this case, obtaining an image associated with a COM object implies obtaining the image from a DLL file over a network, which may be the Internet, a well-known and widely used network. Lastly, for the reasons described above in the rejection for claim 1, it would have been obvious to one of ordinary skill in the art, having the teachings of Oppenheim, Mac OS 8.5, and the COM specification before him at the time the invention was made, to modify the objects taught by Oppenheim and Mac OS 8.5 so that they are implemented as COM objects, meaning that they are provided via a server as described above.

Claims 28, 29 and 31 are rejected under 35 U.S.C. 103(a) as being unpatentable over the combination of Oppenheim and Mac OS 8.5, which is described above, and also over U.S. Patent No. 5,778,227, which is attributed to Jordan. As shown above, the manner in which transformer objects are presented and used, as described by Oppenheim and Mac OS 8.5, teaches the method recited in claim 16 for creating a mapping between a source object having a source object node and a target object having a target object node. Neither Oppenheim nor Mac OS 8.5, however, explicitly discloses displaying a transformer object palette in a graphical user interface; displaying the image associated with a transformer object on this palette; and allowing a user to drag and drop the image associate with a transformer object from the palette to a mapping screen region, as is recited in claim 28.

The U.S. Patent of Jordan discloses a method for efficiently and graphically modifying objects. These objects refer to objects within the context of object-oriented programs (see column3, lines 1-3). Moreover, Jordan discloses that an existing object may be changed by dragging and dropping, onto the object, an icon representing an attribute to add to the object, wherein particularly, the icon is dragged from a palette (see column 9, lines 19-32). Like this icon disclosed by Jordan, the transformer object taught by Oppenheim can also be used to modify an existing object by dragging and dropping the transformer object onto the existing object.

It would therefore have been obvious to one of ordinary skill in the art, having the teachings of Oppenheim, Mac OS 8.5, and Jordan before him at the time the invention was made, to modify the manner of presenting objects taught by Oppenheim and Mac OS 8.5, so that the transformer objects are displayed on a palette in a graphical user interface, like the icons described by Jordan. It is understood that, like the icons described by Jordan, the transformer objects may be dragged and dropped from the palette. It would have been advantageous to one of ordinary skill to utilize such a combination because a palette is an organized, and accepted means for presenting user-selectable functions, as is demonstrated by Jordan.

In regard to claims 29 and 31, the provision of the transformer objects of Oppenheim entails providing a plurality of script components with computer-executable instructions for performing one of a plurality of functions, a plurality of images individually associated with one of the plurality of functions and individually having an input and an output, and wherein the interface component of the transformer object is associated with the plurality of script components and the plurality of graphical components, as was shown above in the rejection for

claim 23. And as is shown above in the rejection for claim 28, the manner of presenting these function objects as disclosed by Oppenheim, as modified by the teachings of Jordan, involves displaying a transformer object palette in the graphical user interface, displaying a least one the images associated with a transformer object on this palette, and allowing a user to drag and drop at least one of the images from the transformer object palette to a mapping region in the graphical user interface. Specifically regarding claim 31, it is further shown above in the rejection for claim 30, that the mapping tool disclosed by Oppenheim necessitates the inclusion of a compiler component, wherein this compiler component is invoked to generate compiled output code. The computer executable instructions from the scripts of the transformer objects are provided to this compiler via the interface of the transformer object, as is shown above.

Claims 36, 39 and 40 are rejected under 35 U.S.C. 103(a) as being unpatentable over the combination of Oppenheim and Mac OS 8.5, which is described above, and also over U.S. Patent No. 6,496,870, which is attributed to Faustini. As shown above, the manner in which transformer objects are presented and used, as described by Oppenheim and Mac OS 8.5, teaches the method recited in claim 33 for creating a function object for use in creating a mapping between a source object having a source object node and a target object having a target object node. Neither Oppenheim nor Mac OS 8.5, however, explicitly discloses the idea expressed in claim 36 for receiving function information from a user, and creating the computer executable instructions based on the function information. More specifically, Oppenheim and Mac OS 8.5 do not disclose the idea expressed in claim 39 for prompting a user for information related to the function object; receiving prompted information from the user via the graphical user interface;

creating the script component having computer-executable instructions for performing a function using the prompted information; and creating the graphical component associated with the function and having an input and an output using the prompted information. Consequently, Oppenheim and Mac OS 8.5 do not disclose the idea that prompting the user for information related to the function object comprises providing a wizard in the graphical user interface, as is expressed in claim 40.

Like Oppenheim, the U.S. Patent of Faustini concerns object-oriented programs, and more particularly, the creation of object-oriented programs. Faustini particularly states:

> *Visual Basic and Visual C++ are examples of two current programming environments with an object oriented foundation that have been developed and made available for programmers to use in creating applications for the Windows 3.x, Windows 95 and Windows NT platforms. While Visual Basic and Visual C++ undoubtedly make program development easier by including tools, called Wizards, that relieve the programmer of the necessity to write the underlying Basic or C++ code needed to create and implement such typical graphical user interface (GUI) elements...* (See column 4, lines 41-53).

The transformation objects disclosed by Oppenheim may be implemented on a Windows NT platform (see column 4, lines 53-57). Moreover, the programs used to create such transformation objects may be written in visual basic or visual C++, as is described above in the rejection for claim 35.

It would therefore have been obvious to one of ordinary skill in the art, having the teachings of Oppenheim, Mac OS 8.5, and Faustini before him at the time the invention was made, to modify the manner of creating transformer objects taught by Oppenheim and Mac OS

8.5, so that the transformer objects may be created using a wizard, as taught by Faustini. In other words, it would have been obvious to modify the manner of creating function objects taught by Oppenheim and Mac OS 8.5, so that it includes receiving function information from a user, and, creating the script of the transformer object to perform the function, based on the information received from the user. It is further understood that using a wizard to create a function object necessitates prompting a user for information related to a transformer object, receiving the prompted information from the user, creating the script component of the transformer object using the prompted information, and creating an image associated with the transformer object. It would have been advantageous to one of ordinary skill to utilize such a combination because wizards relieve the programmer of the necessity to write code, as is taught by Faustini.

## *Conclusion*

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Blaine Basom whose telephone number is (571) 272-4044. The examiner can normally be reached on Monday through Friday, from 8:30 am to 5:30 pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kristine Kincaid can be reached on (571) 272-4063. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

btb
5/1/2006

**TADESSE HAILU**
***Patent Examiner***